TP N° 4 de Réseaux
Etude des protocoles de la couche transport d'Internet
UDP et TCP

Pascal Sicard

1 INTRODUCTION

L'objectif de ce TP est d'observer et de commencer à comprendre le fonctionnement des protocoles de la couche transport d'Internet **UDP** (User Datagram Protocol) et **TCP** (Transmission Control Protocol).

Avant d'entamer les manipulations, voici quelques rappels :

1.1 Architecture des réseaux, protocoles et services

Pour communiquer sur un réseau, les machines utilisent un ensemble de règles et de conventions appelées **PROTOCOLES**. Partant du principe de modularité et compte tenu de leurs complexités, les protocoles ont été structurés **en couches** dans le but de faciliter et de contrôler leurs implémentations.
L'un des avantages de cette structuration est d'isoler les différents protocoles pour que tout changement introduit sur l'un d'eux n'affecte pas les fonctionnalités des autres.
Ce modèle offre des interfaces entre les différentes couches afin de permettre aux protocoles d'une couche donnée d'interagir avec ceux des couches qui lui sont directement adjacents. En effet, chacune des couches s'appuie sur des **services** offerts par une couche inférieure et vise à offrir ses services à la couche qui lui est supérieure.

1.2 Présentation des protocoles étudiés

La famille de protocoles TCP/IP est devenu un standard de fait, de part son utilisation dès le début d'Internet.

La figure 1 montre les diverses interactions qui existent entre les principaux protocoles de la famille de protocoles d'Internet :

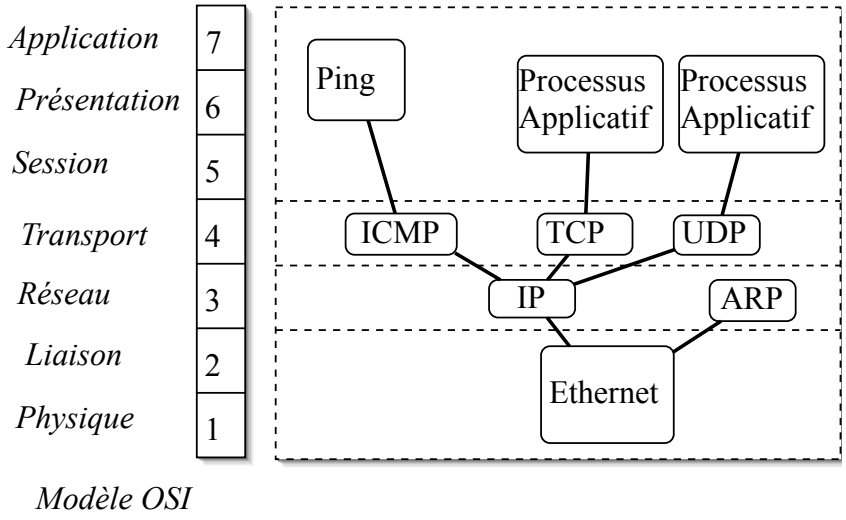

FIG. 1 – Principaux protocoles

Dans la famille de protocoles d'Internet, la couche de niveau transport est la couche la plus haute avec laquelle les processus communiquent pour envoyer ou recevoir des données. Deux protocoles de niveau transport sont définis : TCP et UDP. Du point de vue de l'utilisateur, les qualités de services proposés par ces deux protocoles sont très différentes.

1.2.1 Le protocole TCP

Le service offert par TCP peut être comparé à celui offert par le téléphone : Quand vous téléphonez à quelqu'un, le dialogue ne peut s'instaurer qu'a partir du moment ou votre interlocuteur décroche son combine et dit "ALLO" (ce qui correspond a l'établissement de la connexion) : vous pouvez alors dialoguer...
Un protocole offrant un tel service, en l'occurrence TCP, est dit un protocole **orienté connexion** : cela signifie que si deux processus veulent s'échanger des données par TCP, ils doivent préalablement établir une connexion (dite « virtuelle »). Une fois la connexion établie, TCP garantit que toutes les données envoyées par le premier processus seront reçues sans la moindre erreur par le deuxième : il n'y aura ni perte, ni modification des données. De plus, si les données sont envoyées dans un certain ordre, elles seront réceptionnées dans le même ordre.
Le service proposé par TCP possède aussi la caractéristique d'être de type "**byte-stream**" (flux d'octets). Ainsi si un premier processus envoie 5 puis 15 caractères, ceux-ci peuvent être récupérés de différentes manières par le processus distant : en une lecture de 20 caractères, deux lectures de 10 caractères, deux lectures de 7 et une lecture de 6 caractères...

Autrement dit, il n'y a pas de découpage fixé au niveau applicatif dans le flux de données véhiculées par les paquets TCP.

1.3 Le protocole UDP

Le service offert par le protocole UDP peut être comparé à celui offert par la poste : quand vous postez une lettre, et si vous demandez le tarif habituel, il peut arriver que cette lettre se perde. Par ailleurs, il se peut qu'elle arrive à destination après une autre lettre qui avait pourtant été postée après elle...
De la même manière, deux processus peuvent aussi utiliser le protocole UDP pour s'envoyer des données. Avec UDP, **aucune connexion préalable n'est nécessaire**, mais à l'inverse de TCP, UDP ne donne aucune garantie quant à la qualité du service proposé : des données peuvent être perdues, arrivées dans le désordre, éventuellement modifiées... C'est à l'utilisateur d'effectuer ces contrôles, si cela est nécessaire.
Alors que TCP véhicule un flux d'octets entre les applications, UDP véhicule un flux de paquets de données applicatives : si un premier processus envoie 5 puis 15 caractères par UDP, le processus distant ne pourra pas les lire en une seule fois : il devra lire les deux paquets séparément.

Avec TCP, l'unité d'information est l'octet ; avec UDP c'est le message.

En résumé, deux applications peuvent communiquer en utilisant le protocole TCP ou UDP. Quels peuvent être les critères permettant de choisir l'un plutôt que l'autre ? Les critères les plus évidents sont directement liés aux qualités de services de TCP et UDP. Par exemple, si l'on veut absolument une communication fiable à 100% (par exemple transfert de fichier), on choisira TCP...

1.4 Problème de l'adressage au niveau transport

Dans le TP précédent, vous avez vu qu'une machine était identifiée par une adresse INTERNET ; C'est cette adresse que TCP indique à IP quand il veut envoyer un message, elle se retrouve ensuite dans l'entête IP. Mais au niveau *transport*, cette adresse de niveau *réseau* n'est pas suffisante, que vous vouliez utiliser TCP ou UDP.

A un instant donné, plusieurs applications utilsant TCP ou UDP peuvent tourner en même temps sur une machine, donc plusieurs connexions ont été ouvertes par plusieurs processus sur une même machine ; l'adresse INTERNET de la machine ne permet pas, à elle seule, de distinguer ces connexions ; la notion de ***port*** a été introduite pour faire ces distinctions. On peut parler d'adressage de niveau transport.

Un numéro de port est un entier de 16 bits qui sert à identifier un point d'accès aux protocoles de la couche 4. Ainsi une connexion TCP peut être identifiée de façon unique par deux couples [**adresse INTERNET, numéro de port**] source et destination. Quand un processus veut envoyer un paquet vers un autre processus d'une machine distante, il doit indiquer l'adresse INTERNET de cette machine, et un numéro de port particulier sur celle-ci. La figure 2 résume ces notions pour un petit exemple : elle représente deux connexions de deux clients sur la machine 2 vers un même serveur sur la machine 1.

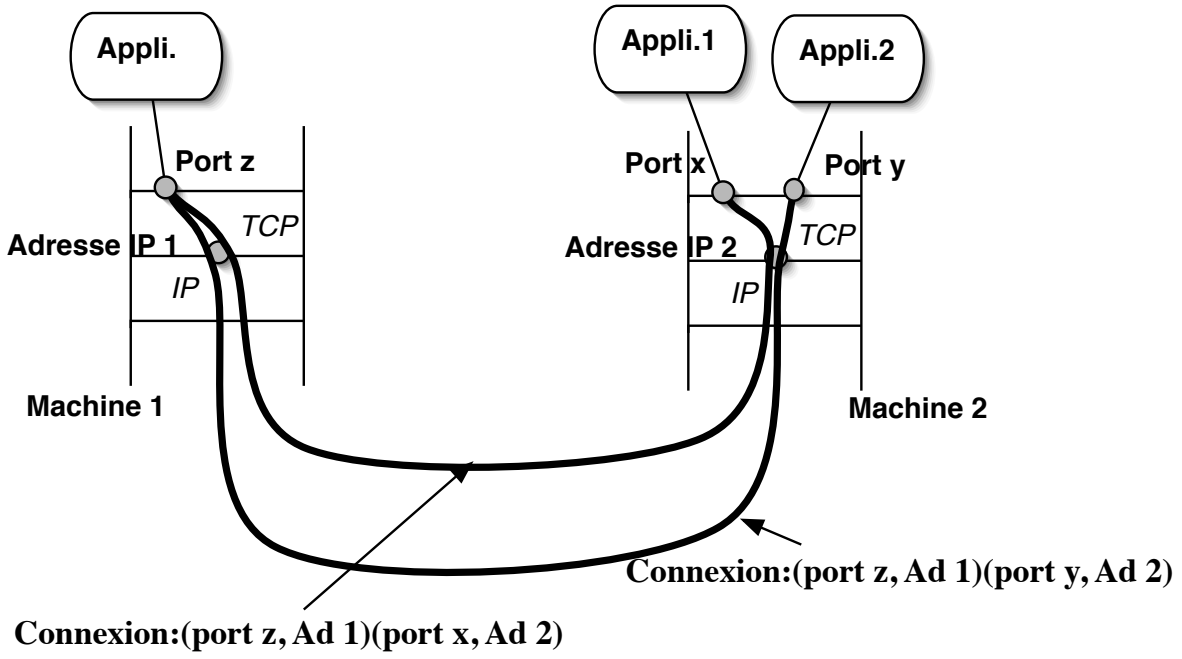

FIG. 2 – Connexions TCP

1.5 Interface logicielle sur les protocoles TCP et UDP

Jusqu'ici, nous vous avons présenté les aspects théoriques de TCP et UDP. Comme ces deux protocoles sont directement accessibles par n'importe quel processus utilisateur (ou applicatif), cela signifie que le système met à la disposition de l'utilisateur un certain nombre de primitives permettant d'envoyer et recevoir des données par TCP ou UDP.
Toutes ces primitives constituent une interface (API : Application Program Interface).
Il existe plusieurs interfaces possibles, selon le type de système d'exploitation et le langage de programmation utilisé. Sous les systèmes UNIX, les deux interfaces les plus utilisées sont :

- **les sockets** BSD (Berkeley Software Distribution).
- Systeme V TLI (Transport Layer Interface).

Dans ce TP, nous utilisons les **sockets**. Le langage de programmation à utiliser est à l'origine le C, mais dans un premier temps, vous ne serez pas amenés à développer en C. Pour étudier les protocoles TCP et UDP, vous utiliserez un utilitaire vous proposant une interface souple et simplifiée sur les sockets : **socklab.** Ce "laboratoire" à socket vous permet en fait d'appeler de façon interactive les primitives de base de manipulation des sockets.
Une **socket** définit aussi un "**point d'accès**" que vous devez créer pour envoyer ou recevoir des données par le protocole UDP ou TCP.

2 DEROULEMENT DU TP :

Dans ce TP, au moins 2 stations doivent être connectées sur un même réseau.

- Les captures et les analyses des paquets générés par les manipulations se feront à l'aide de l'utilitaire **Wireshark** (cf. documentation).
- Ces captures pourront se faire sur l'une quelconques des 2 machines.
- Vous pouvez sauvegarder vos captures en format « Wireshark » ou format ASCII (voir le mode d'emploi dans la documentation « Outils »).
- Une option intéressante de Wireshark pour vous aider se trouve dans le menu **statistics/ flow graph**.

On résumera les captures de paquets à l'aide de croquis temporel faisant apparaître les paquets et les champs intéressants des entêtes. Vous utiliserez le logiciel **socklab** (voir documentation fournie), dont le but est de vous proposer une interface sur les primitives de manipulation des sockets.

2.1 Le protocole UDP

Rappel : La figure 3 donne le format de l'entête UDP.

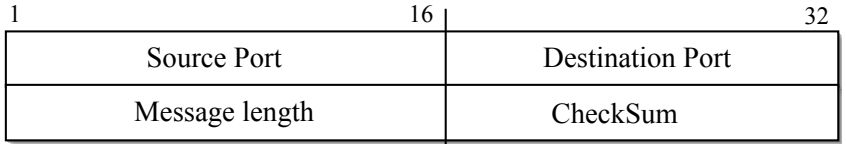

FIG. 3 – Format de l'entête UDP

- Sur deux stations différentes, lancez **socklab** en précisant que vous désirez uniquement travailler avec le protocole UDP :
  # socklab udp
- Sur chacune des deux stations, créez une socket UDP. Ces deux sockets seront utilisées pour échanger des messages entre les deux stations :
  socklab-udp> sock
  Equivalent des commandes en mode standard : **sock udp** (création de socket udp) et **bind** (affectation d'adresse IP et de numéro de port). Voir documentation sur Socklab. **Notez bien le numéro de port qui vous est retourné : il identifie de facon unique la socket sur la machine.**
- Sur une des deux machines, demandez à émettre un paquet de données vers l'autre machine, en précisant son nom et le numéro de port de la socket précédemment créée :
  socklab-udp> sendto <Id de Socket> <nom de machine> <numéro de port>
- Tapez et validez la chaîne de caractères qui doit être transmise vers la machine distante. A quoi sert l'identificateur de **socket** ? Connaît-on le numéro de la socket distante ?

- Sur la deuxième machine, demandez à recevoir un paquet de données sur la socket précédemment créée, en précisant le nombre maximum d'octets à lire :
  socklab-udp> recvfrom <Id de Socket> <nb d'octets>
- Capturez et analysez le (ou les) paquets engendrés par l'émission du message précédent.

- Précisez le rôle de chaque champ de l'entête UDP. Quelles sont les informations passées à IP par UDP à l'émission d'un paquet (données + paramètres de service) ?

Effectuez les variantes suivantes lors de l'échange de données à l'aide d'UDP :

- Demandez la réception avant l'émission des données ;

- Envoyez plusieurs paquets avant de demander leur réception ;

- Croisez l'émission des paquets : sur chacune des stations, demandez d'envoyer un paquet, puis demandez de lire le paquet envoyé par la station distante.

- Notez ce qu'il se passe quand vous demandez à recevoir plus ou moins d'octets que la station distante en envoie.

- Envoyez un paquets vers une machine que vous aurez au préalable, débranchée du réseau. Observez ce qui se passe sur le réseau à l'aide d'une troisième machine (ou sur la machine émettrice).

- Envoyez plusieurs paquets de taille importante vers une machine de façon à "saturer" le récepteur (remplir son buffer de réception). Par exemple envoyer 6 paquets de 5000 octets, puis essayez de les réceptionner (Sous socklab taper **# 5000** à la place du message à envoyer).
  Vous pouvez connaître la taille du buffer de réception à l'aide de la commande ***options*** de Socklab.
- Envoyez un paquet UDP vers un port inexistant.

- Observez les paquets qui sont alors échangés entre les machines. Quel protocole est utilisé alors ? Expliquez ce qui se passe entre les différents protocoles sur la machine réceptrice lors de la réception du paquet UDP.

Expliquez toutes ces observations, écrivez un résumé sur le fonctionnement du protocole UDP.

2.2 Le protocole TCP

Rappel : La figure 4 donne le format de l'entête TCP.

Signification des différents champs :

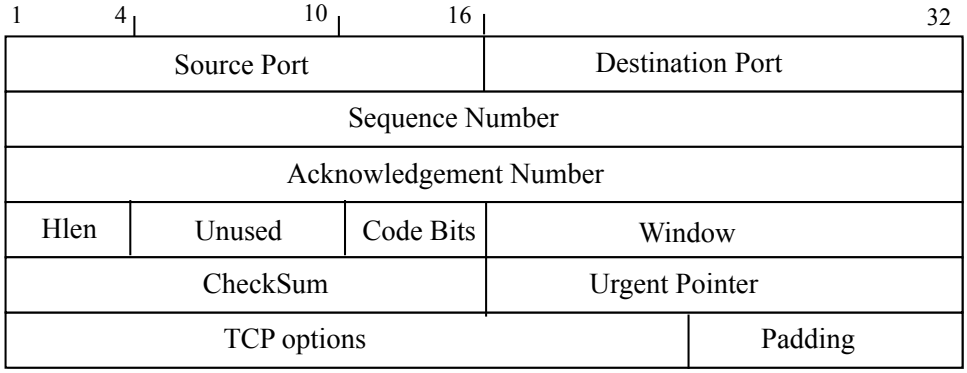

FIG. 4 – Format de l'entête TCP

- **Source port et destination port** ont le même sens que dans les paquets UDP.
- **SEQUENCE NUMBER et ACK NUMBER** sont utilisés pour le séquencement et la récupération d'erreurs des données (le flag ACK indique si le champ ACK NUMBER contient une valeur valide).
- **HLEN** indique la taille de l'entête TCP en mots de 4 octets (la taille de l'entête TCP est variable : elle peut être complétée par une ou plusieurs options de 4 octets chacune).
- **CHECKSUM** a le même sens que dans les paquets UDP.
- **URGENT POINTER** est utilisé pour le transport de "données urgentes" (le flag URG indique si le champ URGENT POINTER contient une valeur valide).
- **Le champ Code bits** est composé des six flags suivants : **URG, ACK, PSH, RST, SYN, FIN.**
  - Les flags **SYN et FIN** sont utilisés pour l'établissement et la fermeture des connexions virtuelles.
  - Le flag **RST** est utilisé pour signaler au destinataire une demande de re-initialisation des connexions qui sont dans un état incertain (SYN dupliqués, fermeture anormale, panne ...).
  - Le flag **PSH** ne sera pas étudié.
- Le champ **WINDOW** est utilisé pour le contrôle de flux.

2.2.1 Etablissement d'une connexion

Sur deux machines, lancez **socklab** en précisant que vous désirez uniquement travailler avec le protocole TCP : # **socklab tcp**

- Sur la première machine, créez une socket "**passive**". Notez le numéro de port de la socket ainsi créée, puis mettez-la en attente de connexion.
  socklab-tcp> passive
  Equivalent des commandes en mode standard sock tcp (création de socket), bind (affectation d'adresse IP et de numéro de port) et listen (attente) sur cette socket.
  socklab-tcp> accept
- Sur la deuxième machine, créez une socket "**active**", et connectez-la sur la machine et le port de la socket passive précédemment créée. Ceci peut se faire grâce à la commande :
  socklab-tcp> connect <nom de machine> <numero de port>
  Equivalent des commandes en mode standard : ***sock tcp*** *(création de socket),* ***bind*** *(affectation d'adresse IP et de numéro de port locaux) et* ***connect*** *(requête de connexion) sur cette socket.*
- Pourquoi la première socket créée est dite "**passive**", et la seconde "**active**" ?
- Quels sont les rôles respectifs de ces deux sockets dans l'établissement de la connexion ?
- Analysez les paquets générés lors de l'établissement de la connexion.

- Décomposez les étapes de cette connexion : enchaînement dans le temps des demandes de services à TCP et des messages échangés.
- Quel est le rôle du flag SYN ?

- Quelles sont les informations échangées durant ces étapes ? A quoi servent les numéros de séquence et d'acquittement ? (attention les numéros donnés par Wireshark sont « relatifs », voir les vrais dans l'hexadécimal)

- A quoi servent chacune des options utilisées ici par TCP ?
- Expliquez ce qu'il se passe au moment de l'"*accept*".
- Essayez de le faire avant et après le "*connect*".
- Regardez la liste des sockets (commande status).
- Ouvrez plusieurs connexions d'une machine vers un même port destinataire. Observez les connexions sous *socklab* à l'aide de la commande **status** sur le serveur et sur le client.
- Qu'est ce qui identifie réellement une connexion, c'est-à-dire, comment TCP associe les messages reçus aux différentes connexions en cours ?
- Faites une demande de connexion (**Connect**) vers un port inexistant. Expliquez ce qu'il se passe.

2.2.2 Etude du séquencement et de la récupération d'erreur

- Etablissez une connexion par sockets TCP entre deux machines, grâce aux commandes vues précédemment.

- Echangez des données entre les deux machines grâce aux deux commandes **read** et **write** (à la place d'un message normal, vous pouvez utiliser la notation **#nnn** pour envoyer un message de *nnn* octets). Essayez en particulier avec une taille de donnée importante (5000 octets par exemple).

- Analysez les paquets engendrés par le transport des données. Expliquez le rôle des champs SEQUENCE NUMBER et ACK NUMBER dans l'entête des paquets TCP.

- Il y a t-il toujours un acquittement par paquet de donnée ? Pourquoi ?
- Envoyez un message vers une machine que vous aurez au préalable débranchée du réseau.

- Observez ce qui se passe en capturant les paquets sur la machine émettrice. Expliquez en détail les mécanismes utilisés dans ce genre de cas (perte d'un message), comparez avec le protocole UDP.

- Buffer d'émission de la récupération d'erreur :
  - Modifiez la taille du buffer d'émission à 1000 octets sur une machine sur laquelle vous avez créé une socket passive (commande socklab *option* sur la socket avant l'ouverture d'une connexion).
  - Ouvrez une connexion vers cette socket.
  - Envoyez 10 000 octets depuis la machine sur laquelle le buffer d'émission a été réduit.
  - Observez les paquets circulant à ce moment-là.
  - Quelle influence la taille du buffer d'émission a t-elle ?
  - Rappelez la fonctionnalité du buffer d'émission de TCP.

- Donnez de façon informelle l'algorithme de mise à jour de ces deux champs. On fera bien attention à distinguer le cas où l'on reçoit un paquet et le cas ou l'on doit émettre un paquet. On pensera aux cas de pertes.

2.2.3 Contrôle de flux

- Ouvrez une connexion TCP.

- Emettez sur cette connexion un message de taille supérieure à celle du buffer de réception (voir les options de la socket).
- Remarque : On peut changer la taille du buffer de réception mais seulement avant l'ouverture de la connexion (côté serveur sur la socket passive).

- Analysez les paquets échangés ? Regardez en particulier le champ ***Window*** de l'entête TCP.
- Faites ensuite des *read* successifs coté récepteur de 5000 octets.

- Analysez les derniers paquets échangés. Regardez en particulier le champ ***Window*** de l'entête TCP. Expliquez.

- Refaîtes l'expérience en libérant le buffer du récepteur par des *reads* de peu d'octets. A partir de quel moment l'émetteur est-il « débloqué » ? Pourquoi ?

2.2.4 Libération d'une connexion

- Après ouverture d'une connexion entre deux machines, fermez la connexion : **close**
- Analysez les paquets générés lors de la fermeture de la connexion de chaque côté. Décomposez les étapes de cette fermeture. Quel est le rôle du flag FIN ?

- Résumez les échanges de messages en spécifiant la valeur des champs spécifiques à cette phase de la communication.
- Après fermeture d'un seul des deux côtés, essayez de continuer à émettre de l'autre côté (par un *write*). Que se passe t-il (observez les paquets échangés). Si vous refaites un *write* que se passe t-il ? pourquoi ?

- Essayez différents scénarios de fermeture à l'aide de la commande **shutdown** (voir documentation Socklab).
  Par exemple, réaliser une fermeture en sortie (**shutdown out**) sur une machine, puis une émission de données depuis l'autre.
  Peut-on continuer à lire les données envoyées ? Peut-on continuer à faire un write après un **shutdown out**.
  Faites de même pour le **shutdown in** et **both**.
- Expliquez les avantages et inconvénients de ces différents types de fermeture (*close* et *shutdown*).
- Résumez le fonctionnement de TCP en ce qui concerne l'ouverture et la fermeture de connexion grâce à un automate dont les entrées sont les commandes des sockets (CONNECT, WRITE...) ou l'arrivée de messages particuliers (DATA, SYN, ACK...), et les sorties sont les envois de messages. Un automate de Mealy semble plus approprié (voir l'ébauche sur la figure 5).

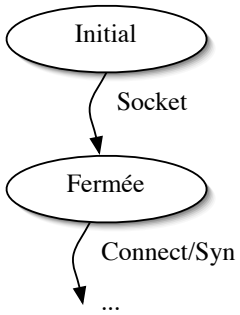

FIG. 5 – Automate ouverture de connexion TCP pour une socket active

Dessinez le graphe de l'automate pour une socket active et un pour une socket dite passive. Commentez cet automate en rappelant les expérimentations effectuées.

2.3 Exercices de synthèse (A faire attentivement)

- Observation de la commande talk

Faites un échange de caractères entre deux machines à l'aide de l'utilitaire **talk** (talk

<nom utilisateur>@<nom de machine officiel>).

- Capturez sur une troisième machine les messages échangés lors de ce dialogue.
- Analysez les paquets capturés. Expliquez ce qu'il se passe au niveau du réseau (protocole transport utilisé, ouverture de connexion, échange des caractères, fermeture de connexion, ...).

- Observation de la commande rlogin

Comme précédemment, sur une des machines, capturez les paquets circulant sur le réseau.

- Faites ensuite un ***rlogin*** d'une deuxième machine sur une troisième (compte ***guest***, mot de passe ***guest./***).
- Puis procédez à une ou deux commandes Unix à travers ce login.
- Enumérez les messages échangés lors de cette manipulation et expliquez ce qui se passe au niveau du réseau.
- Comment les commandes sont elles envoyées à destination ?
- Pourquoi les caractères des commandes sont ils renvoyés à la source ?

Regardez le contenu du fichier /etc/services et retrouvez les numéros de port des serveurs talk et rlogin

---

Références Bibliographiques :

- RESEAUX 4ème Edition Andrew Tanenbaum - InterEditions
- Réseaux locaux et Internet (des protocoles à l'interconnexion) Laurent Toutain – 2ème Edition - HERMES
- Analyse struturée des Réseaux Des Applications de l'Internet aux infrastructures de télécommunication
  James Kurose et Keith Ross
  2^e Edition - Pearson Education
- TCP/IP : Architecture, protocoles, applications Douglas COMER, InterEditions, 1992

Très techniques sur les protocoles TCP/IP :

- Principles, Protocols and Architecture - Volume I Design Implementation and Internals Douglas E. Comer, David L. Stevens
- Internetworking with TCP/IP - Volume II Design Implementation and Internals Douglas E. Comer, David L. Stevens